Building a Pragmatic Data Platform with dbt and Snowflake

Appendix 1
Implementing PDP Governance with Snowflake and dbt

Roberto Zagni and Jakob Brandel

Technics Publications
SEDONA, ARIZONA

Introduction

In Chapter 3 of *Building a Pragmatic Data Platform*, we define Governance through five key pillars: Compliance, Security, Privacy, Data Quality, and Access.

While the book discusses the philosophy of governance, this guide serves as a technical reference.

Governance features in the Pragmatic Data Platform

In the "Pragmatic Data Platform" (PDP), we reject the notion that Data Governance is a bureaucratic layer of paperwork that slows down development. Instead, we view Governance as **engineered reliability**.

By combining the infrastructure capabilities of **Snowflake**, the workflow enforcement of **dbt**, and the architectural patterns of the **PDP**, we can implement ***governance as code***. This means policies are versioned, access is automated, and quality is tested in the pipeline, not inspected after the fact.

This Appendix details exactly how the Pragmatic Data Platform (PDP) leverages specific features within Snowflake and dbt Cloud to satisfy the requirements and implement the five governance pillars: **Compliance, Security, Privacy, Data Quality, and Access**.

---

1. Compliance

Goal: Ensure the platform abides by regulations (GDPR, CCPA, SOX) and minimizes legal risk through auditability.

Compliance requires proving *who* did *what* to the data, ensuring that data retention policies are met, and guaranteeing the system is auditable.

A. Snowflake Features

- **Reporting and Monitoring:** Snowflake provides extensive internal tables (like `LOGIN_HISTORY`, `QUERY_HISTORY` in the `ACCOUNT_USAGE` schema) to track system activity.

  This allows you to identify potential non-compliance risks, such as users logging in from non-compliant locations or executing queries on restricted data.

- **Lineage:** Built-in lineage tracking (via `ACCESS_HISTORY`) ensures auditability.

  Unlike simple logs, Snowflake tracks exactly which columns were read. If a user queries `SELECT * FROM customers`, Snowflake records that `email`, `address`, and `phone` were accessed.

- **Business Continuity:** Features like failover, replication, and disaster recovery mechanisms ensure data availability and compliance with regulatory uptime requirements.

- **Certifications:** Snowflake supports compliance with industry standards such as FedRAMP, Cyber Essentials Plus, PCI-DSS, and SOC2.

  This reduces legal and financial risks by inheriting the security posture of the cloud provider. You secure the access; Snowflake secures the hard drives.

B. dbt Features

- **Version Control:** Tight integration with Git allows every change to be versioned.

  This ensures total traceability. If an auditor asks, *"Why did the revenue calculation change on March 1st?"*, you don't ask people; you look at the Git Blame on the model file.

- **Lineage:** Automated lineage tracking in dbt Cloud visualizes data flow.

  This proves that transformations align with compliance requirements. For example, you can visually prove that PII data from the source does not flow into public reporting marts without passing through a hashing transformation.

- **Access and Run Audits:** Detailed logs for job execution and user activity support auditing needs.

- **Exposures:** Documenting external dependencies provides transparency and supports compliance.

  By defining `exposures` in your `yaml` files, you complement the dbt online documentation. It allows you to declare, *"This dashboard is used for Regulatory Reporting,"* ensuring developers know the compliance weight of the upstream models.

C. PDP Methodology

- **Ingest All Data:** The PDP's philosophy of ingesting all data ensures comprehensive historical records.

  We do not filter columns at the ingestion point. This enables forensic analysis when needed—if a regulation changes, you already have the raw data required to re-calculate compliance metrics.

- **Auditable History:** Historical tables store unaltered versions of data (Insert-Only).

  By using Data Vault-style or Snapshot-style tables where records are never updated, only appended, we satisfy compliance auditing requirements. This makes it possible to reproduce historical reports exactly as they looked two years ago.

---

2. Security

Goal: Authenticate users, strictly control entry to the platform, and protect operational integrity.

The biggest security risk in data platforms is usually internal: developers having too much access to production data, or credentials being shared insecurely.

A. Snowflake Features

- **RBAC (Role-Based Access Control):** Enforces fine-grained permissions at the database, schema, and table levels.

  We strictly follow a hierarchy where permissions are assigned to Roles, never to Users.

- **MFA (Multi-factor Authentication):** Provides an extra layer of protection, ensuring only intended people can access your system.

  It is applied by default for users. For automated systems (Service Users), we utilize **Key Pair Authentication** to avoid static passwords.

- **Row-Level Security (RLS):** Ensures precise access control based on user roles and conditions.

  This allows a "Regional Manager" to query the `Sales` table but only see rows where `Region = 'EMEA'`.

- **Continuous Monitoring:** Ongoing threat detection and incident response capabilities integrated with third-party audits.

B. dbt Cloud Features

- **RBAC for Projects:** Role-based access ensures that only authorized users can view or modify specific models, run jobs, or edit environments.

- **User Management:** Enforces separation between personal accounts for developers in DEV environments and service accounts for deployments.

  This is critical. Humans have permissions to write to `dbt_dev_name`. Only the **CI/CD Service Account** has permissions to write to `PROD`.

- **Exposures:** Clear mapping of data dependencies and interfaces for secure and organized workflows.

  By mapping exposures, security teams can see which downstream external systems (Reverse ETL, Dashboards) are consuming sensitive data.

- **Protection of Historical Tables:** Ensures that historical tables are not deleted by doing a full refresh.

  Utilizing the `protected: true` config in dbt guarantees that your history tables stay safe from accidental `dbt run --full-refresh` commands, preserving the security of your audit trail.

C. PDP Methodology

- **Role** Organization**:** Defines clear roles and access levels for all stakeholders (e.g., `TRANSFORMER`, `REPORTER`, `DEVELOPER`).
- **Environment Separation:** Rigid separation between DEV, CI/QA, and PROD environments ensures secure and isolated operations.
- **Service Accounts:** Deployment to CI/QA and PROD is managed exclusively by service accounts.

  This ensures that environments reflect the code in the repository, and data is safe from "cowboy coding" or manual hot-fixes.

- **Data Mart Access:** Access to data marts is restricted based on roles, ensuring secure usage (e.g., only the BI tool service account can read the Gold layer).

---

3. Privacy

Goal: Control visibility. Even if you are allowed in the room (Security), you can't read every paper (Privacy).

Data Science and Analyst teams need real data to work, but they rarely need to see plain-text Social Security Numbers, emails, or credit card details.

A. Snowflake Features

- **Industry Standards:** Compliance with ISO 27001, SOC 2, and HIPAA to meet established security benchmarks.

- **Column-Level Masking:** Sensitive fields can be masked dynamically based on the viewer's role.

```
-- Example Policy
CREATE MASKING POLICY email_mask AS (val string) returns string ->
  CASE
    WHEN current_role() IN ('HR_MANAGER') THEN val
    ELSE '***MASKED***'
  END;
```

- **Data Clean Rooms:** Enable secure and collaborative data sharing.

  This allows you to join your data with a partner's data without either party seeing the other's raw row-level details.

- **Differential Privacy:** Techniques to add statistical “noise” to protect sensitive data during analysis.

B. dbt Cloud Features

- **Model Access Levels:** Permissions at the model level restrict who can access and modify specific datasets.

  Organizing models into separate groups or folders allows you to lock down "Sensitive" folders so that only specific developers can even see the logic.

- **Model Granting:** Models are explicitly granted to roles, ensuring controlled access.

  Using post-hooks or grant configurations in `dbt_project.yml` ensures that privacy rules are applied every time the table is rebuilt.

C. PDP Methodology

- **PDP Governance:** Our default setup adopts best practices to provide an out-of-the-box data platform with a high level of security.

- **PII Framework:** A simple yet complete framework to properly handle PII and sensitive data.

  We can use dbt `meta` tags to identify PII columns in the staging layer. We then apply masking policies immediately, ensuring PII is protected before it spreads to downstream models.

4. Data Quality

Goal: Ensure information is accurate, consistent, easy to find, and reliable.

Bad data erodes trust faster than slow data. The platform must proactively detect issues before they impact business decisions.

A. Snowflake Features

- **Dashboard and Statistics:** Snowflake's Snowsight provides data engineers with an interactive interface to detect data quality issues.

  Visual dashboards help engineers monitor anomalies, outliers, data freshness, and data duplication without needing external tools.

- **Data Discovery:** Snowflake Horizon enables comprehensive data discovery by cataloging datasets across the organization.

  This makes it easier to locate and validate data assets for quality assessments.

- **Data Observability:** Query and Load history tables, along with the information schema, help data engineers monitor pipeline performance and address quality issues.

- **Time Travel:** Enables rollback to previous states, ensuring data consistency and recoverability.

  If a bad transformation corrupts a table, you can execute `SELECT * FROM table AT(OFFSET => -60*5)` to restore the data as it was 5 minutes ago.

B. dbt Cloud Features

- **Testing:** Includes built-in tests for schema and data integrity (unique, not null, accepted values).

- **Personalized Tests:** Developers can write custom tests to enforce unique business rules.

  For example, a SQL test ensuring that `order_date` is never later than `shipping_date`.

- **Model Versioning:** Versioned models ensure consistent outputs and business continuity.

  This supports incompatible structural changes (breaking changes) by allowing legacy consumers to keep using v1 while new consumers migrate to v2.

- **Data Contracts:** Define and enforce expectations for datasets, reducing risks of downstream failures.

  Contracts catch schema changes from upstream sources *before* they break the pipeline.

- **dbt Explorer:** Provides a visual map of data models, aiding in quality assurance and impact analysis.

C. PDP Methodology

- **PDP Layered Structure:** Leverage patterns to simplify development and ensure consistent data handling across different layers (Ingestion, Storage, Refined, Delivery).
- **Insert-Only Models:** Ensures that data changes are tracked and auditable.
- **Testing Guidelines:** Standardized practices for testing for each layer.

  We enforce freshness tests on Sources, key integrity tests on Storage, and business logic tests on Delivery.

- **Documentation:** Clear guidelines for documenting models, tests, and data lineage to support data quality efforts.
- **Way of Working:** Ensuring that the code is reviewed by colleagues, also known as the **four eyes principle**, ensures data quality and operational continuity.

---

5. Access

Goal: Ensure data is available, scalable, and easy to use, while maintaining strict controls.

Data is useless if it is locked away in a format tools can't read, or if the database is too slow to query during peak hours.

A. Snowflake Features

- **Replication:** Seamless replication across regions ensures data is always available.
- **Multi-Cloud:** Operates in multi-cloud environments for flexibility and redundancy.

- **Business Continuity:** Features such as failover, replication, and disaster recovery ensure data availability.
- **Separation of Storage and Execution**: Optimizes performance and cost.

  Heavy data ingestion jobs run on a `LOADING_WH`, ensuring they never slow down the dashboard queries running on the `REPORTING_WH`.
- **Autoscaling:** Automatically adjusts resources based on demand, maintaining availability during peak times.
- **Tagging and Classification:** Tools for tagging and classifying data; ensure that sensitive data is managed effectively and easy to search.

B. dbt Cloud Features

- **CI/CD Deployment Jobs:** Automates deployments, ensuring consistent releases.
- **Alerting and Scheduling:** Proactive monitoring for scheduled jobs ensures users know immediately if data is stale.
- **Basic Orchestration:** Simplifies dependency management and execution order, ensuring data is built in the correct sequence.

C. PDP Methodology

- **Interface-Based Organization:** Modules and layers are built around clear interfaces for interoperability.

  We treat the Delivery Layer as a public API (Interoperability), promising backward compatibility to downstream users.
- **Testing and Documentation:** Comprehensive test and documentation coverage ensure access remains reliable and secure.

Conclusion

Governance as an Enabler, Not a Blocker

Traditional data governance often feels like a brake system—a set of bureaucratic hurdles, approval committees, and manual checks designed to slow down development in the name of safety.

In the Pragmatic Data Platform, we flip this script. We believe that **speed and safety are not opposites; they are partners.** Just as better brakes allow a racing car to drive faster, better automated governance allows data teams to deploy more frequently.

By baking compliance, security, and quality directly into the Snowflake infrastructure and the dbt codebase, we move from a model of "Gatekeepers" (humans reviewing spreadsheets) to a model of "Guardrails" (automated tests and policies).

As you implement the patterns detailed in this guide—from the "Insert-Only" storage layer to the "Four Eyes" code review principle—remember that your goal is not just to check a box for an auditor. Your goal is to build a platform that the business trusts implicitly. When stakeholders know that privacy is enforced by default and quality is guaranteed by contract, they stop questioning the data and start using it to drive decisions.

That is the ultimate promise of the Pragmatic Data Platform: a system where doing the *right* thing is also the *easiest* thing.